Laboratorul 2

Sistemul de fișiere

1 Ierarhie

În sistemele de operare de tip UNIX fișierele și directoarele sunt organizate într-o structură arborescentă. Rădăcina este notată cu `/` și se mai numește și *root*. În rădăcină se găsesc mai multe directoare și fișiere. La rândul lor, aceste directoare pot conține alte fișiere și directoare. Directoarele dintr-un director se mai numesc și subdirectoare sublinind relația ierarhică.

De exemplu, mesaje de sistem se găsesc în fișierul `/var/log/messages`. Acesta se află în directorul `log` care se află la rândul său în directorul `var` care se află în rădăcină. Alt exemplu este directorul personal de lucru `uso` care conține două directoare, care la rândul lor au câte un subdirector în care se găsesc diferite fișiere.

```
$ tree uso
uso
|-- curs
|   `-- 1
|       |-- uso-curs-1.pdf
|       `-- uso-curs-1.tex
|-- lab
|   `-- 1
|       |-- uso-lab-1.aux
|       |-- uso-lab-1.fdb_latexmk
|       |-- uso-lab-1.fls
|       |-- uso-lab-1.log
|       |-- uso-lab-1.pdf
|       |-- uso-lab-1.synctex.gz
|       `-- uso-lab-1.tex
`-- plan

4 directories, 10 files
```

2 Linia de comandă

Majoritatea comenzilor vor fi executate în cadrul unui terminal (ex. `xterm(1)`, GNOME Terminal etc.). Deși dificil și aparent mai complicat la început, folosirea terminalului oferă multe avantaje precum flexibilitate, automatizarea sarcinilor și control la distanță al altor mașini.

Un terminal tipic are la bază un program de tip `shell` (ex. `bash(1)`, `ksh`, `zsh`) care gestionează și execută comenzi secvențial sau în paralel. Promptul unui terminal indică faptul că se așteaptă o comandă de la utilizator și este în cea mai simplă formă constituit din simbolul `$` sau `%` pentru utilizatorii comuni și `#` pentru administrator (denumit `root` în UNIX). În Linux se practică un prompt mai elaborat care poate conține numele utilizatorului, numele mașinii, și/sau directorul curent. De exemplu, promptul `root@lab uso#` indică faptul că utilizatorul `root` este logat pe terminalul de pe mașina `lab` și se află în directorul `uso`.

Tot ce este scris în dreapta promptului reprezintă comanda utilizatorului către mașină.

```
$ echo "Hello, World!"
Hello, World!
$
```

În exemplul de mai sus a fost executată comanda echo cu parametrul `"Hello, World!"`. Rezultatul comenzi, dacă există, este afișat fără a fi prefixat cu prompt. Încheierea comenzii este semnalată prin reapariția promptului.

De fiecare dată când vedeți o comandă necunoscută citiți manualul pentru a afla ce face

```
$ man echo
```

Din motive istorice, manualul sistemului de operare (ce include manualele comenzilor) este împărțit în secțiuni. Astfel pot exista mai multe intrări cu același nume dar în secțiuni diferite. Vezi cunoscuta funcție `printf`.

```
$ man printf
$ man 3 printf
```

Prima instrucțiune s-ar putea să vă surpindă afișând manualul comenzii `printf` nu a funcției C `printf`. Comenzile de terminal se află de regulă în secțiunea 1, pe când funcțiile se află în secțiunea 3. Pentru a accesa manualul funcției trebuie să specificăm un argument în plus comenzii `man(1)` care specifică secțiunea explicit. Din această cauză când ne referim la o comandă sau o funcție punem la sfârșit și secțiunea din manual în care este documentată: `printf(1)` versus `printf(3)`. Este bine de știut că există manual și pentru comanda de citit manuale

```
$ man man
```

3 Navigare

În general, fiecare utilizator are un spațiu de lucru propriu în care își poate desfășura activitatea. Acest spațiu este găzduit într-un director, de regulă `/home/username`, care este memorat în variabila `$HOME`

```
$ echo $HOME
/home/paul
```

Acest tip de variabilă se mai numește și *variabilă de mediu*. Ele sunt definite dinamic de sistem sau utilizator pentru a fi folosite de programe la execuție. Variabilele sunt în general scrise cu litere mari și precedate de simbolul `$`. O altă variabilă importantă este cea în care sunt memorate căile din sistemul de fișiere în care se găsesc executabilele.

```
$ echo $PATH
/bin:/sbin:/usr/bin:/usr/sbin:/usr/X11R6/bin:
/usr/local/bin:/usr/local/sbin
```

Când este pornit un terminal, acesta de regulă vă plasează în directorul `$HOME`. Folosiți comanda `pwd(1)` pentru a verifica în orice moment unde vă aflați și comanda `ls(1)` pentru a lista conținutul directorului curent.

```
$ pwd
/home/paul/wrk/ub/uso/lab/1
$ ls
uso-lab-1.aux                uso-lab-1.log
uso-lab-1.fdb_latexmk        uso-lab-1.pdf
uso-lab-1.fls                uso-lab-1.tex
```

Toate comenzile se execută relativ la directorul curent: `ls(1)` verifică implicit directorul curent și listează conținutul său.

Dacă doriți să schimbați directorul curent folosiți comanda `cd(1)`. Aceasta primește ca parametru viitorul director curent. El poate fi dat relativ la directorul curent sau în formă absolută pornind de la rădăcină. În aproape toate sistemele de operare `.` simbolizează directorul curent și `..` directorul părinte. Deci dacă vrem din exemplul anterior să ajungem acasă putem folosi oricare dintre următoarele comenzi

```
$ cd ../../../../../
$ cd /home/paul
$ cd $HOME
$ cd
```

Implicit `cd(1)` fără argumente schimbă directorul în directorul `$HOME`.

Pentru a afla unde se află executabilul aferent unei comenzi folosiți comanda `which(1)`

```
$ which ls
/bin/ls
```

Observați că `ls(1)` se află într-unul din directoarele conținute în `$PATH`.

4 Citire și scriere

Pentru a crea fișiere noi text se poate folosi `echo(1)`

```
$ echo "lorem ipsum" > foo
```

unde operatorul `>` redirecționează ieșirea comenzi către fișierul `foo`. Dacă `foo` există, va fi suprascris. Pentru a adăuga la sfârșitul unui fișier existent folosiți `>>`. Fișierele text scurte pot fi rapid afișate în terminal cu ajutorul comenzii `cat(1)`.

```
$ cat foo
lorem ipsum
```

Deși se poate aplica aceiași comandă asupra fișierelor binare, precum executabilele, nu este recomandat deoarece anumite "caractere" rezultate pot fi interpretate de shell drept caractere de control care vor da peste cap funcționarea normală a terminalului. De aceea se folosește utilitarul `hexdump(1)`

```
$ hexdump –C /bin/ls
```

unde opțiunea `-C` indică modul canonic de afișare a binarelor: în stânga octeții în format hexadecimal și în dreapta octeții în format ASCII. Rezultatul este destul de lung și depășește lungimea terminalului. Pentru a parcurge toată informația se recomandă folosirea unui *pager* precum `less(1)`

```
$ hexdump –C /bin/ls | less
```

unde operatorul `|` se numește *pipe*. Un pipe transformă iesirea programului din stânga în intrarea celui din dreapta. Pentru a ieși din `less(1)` apăsați tasta `q`. Pentru a căuta un text folosiți comanda `/`. De exemplu `/print` va căuta șirul de caractere `print`. Evident, `less(1)` poate fi folosit direct pentru a inspecta fișiere și este util mai ales pentru fișiere text mari

```
$ less /etc/passwd
```

Căutarea unui fragment de text într-un fișier se poate face folosind *expresii regulate* în cadrul comenzii `grep(1)`. Fie următorul fișier text

```
$ echo -e "words\nrat\nrata\nratat\nratata\nratatata" > rat
```

unde `\n` reprezintă o linie nouă (ca în limbajul C). Pentru a găsi cuvintele ce conțin `at` în fișier folosim comanda

```
$ grep at rat
rat
rata
ratat
ratata
ratatata
```

Dacă vrem să găsim cuvintele care se termină în litera `t` atunci vom folosi comanda

```
$ grep t$ rat
rat
ratat
```

unde caracterul `$` simbolizează sfârșitul linei. Pentru începutul linei se folosește `^`

```
$ grep ^w rat
words
```

Alte caractere speciale utile, numite *wildcards*, sunt:

- `*` – găsește de 0 sau mai multe ori atomul precedent
- `+` – găsește de 1 sau mai multe ori atomul precedent
- `?` – găsește de 0 sau 1 ori atomul precedent

unde un atom este implicit caracterul precedent sau grupul precedent de caractere. Gruparea mai multor caractere se face cu ajutorul parantezelor rotunde `()`. În exemplul nostru putem căuta toate cuvintele care încep cu `rata` și sunt urmate de `ta` de 0 sau mai multe ori astfel

```
$ grep -E 'rata(ta)*' rat
rata
ratat
ratata
ratatata
$ grep -E 'rata(ta)+' rat
ratata
ratatata
$ grep -E 'rata(ta)?' rat
rata
ratat
ratata
ratatata
```

Observați că am folosit parantezele pentru grupare și opțiunea `-E` pentru a folosi expresii regulate.

5 Redirectarea intrarii si a iesirii comenzilor

In cele ce urmeaza se va folosi comanda *cat* de mai sus pentru ca exersa redirectarea intrarii si iesirii si iesirii comenzilor asa cum au fost prezentate la curs. Multe comenzi Unix, inclusiv *cat*, accepta "-" ca parametru pentru a specifica *stdin*.

```
$  cat - > out
aici introduceti textul de la tastatura
```

```
urmat de ctrl-d care reprezinta EOF
dupa EOF, textul va fi redirectat catre fisierul out


$ cat out
aici introduceti textul de la tastatura
urmat de ctrl-d care reprezinta EOF
dupa EOF, textul va fi redirectat catre fisierul out


$  cat << MY-OWN-EOF >> out
> acelasi exercitiu care foloseste un marker separat
> pentru EOF, respectiv stringul "MY-OWN-EOF"
> textul introdus de la tastatura va fi adaugat
> la sfarsitul fisierului out (operatie de append)
>  MY-OWN-EOF
> MY-OWN-EOF

$ cat out
aici introduceti textul de la tastatura
urmat de ctrl-d care reprezinta EOF
dupa EOF, textul va fi redirectat catre fisierul out


acelasi exercitiu care foloseste un marker separat
pentru EOF, respectiv stringul "MY-OWN-EOF"
textul introdus de la tastatura va fi adaugat
la sfarsitul fisierului out (operatie de append)
 MY-OWN-EOF

$  cat < out
aici introduceti textul de la tastatura
urmat de ctrl-d care reprezinta EOF
dupa EOF, textul va fi redirectat catre fisierul out


acelasi exercitiu care foloseste un marker separat
pentru EOF, respectiv stringul "MY-OWN-EOF"
textul introdus de la tastatura va fi adaugat
la sfarsitul fisierului out (operatie de append)
 MY-OWN-EOF
$
```

Observati ca atunci cand va definiti propriul EOF e necesar ca acest marker sa fie introdus singur pe o line chiar de la inceputul liniei. De asemenea observati un aspect tipic pentru Unix, posibilitatea de a executa acelasi task in mai

multe feluri, de la folosirea propriului marker EOF pana la afisarea continutului fisierului *out*, fie simplu *cat out*, fie redirectand intrarea din fisierul *out* si mizand pe comportamentul standard al comenzii *cat* care scrie pe ecran, asa cum apare in comanda *cat* < *out*.

6 Manipulare

Pentru a schimba valoarea unei variabile se folosește comanda `export`. Aceasta este o comandă tipică shell-ului folosit și găsiți detalii despre ea în manualul shell-ului (ex. `bash(1)`).

```
$ echo $PS1
$
$ export PS1="uso$ "
uso$ echo $PS1
uso$
```

Putem refolosi variabila curentă și doar adăuga informații la început folosind

```
$ export PS1="uso$PS1"
uso$ echo $PS1
uso$
```

unde variabila `$PS1` din dreapta a fost întâi evaluată, iar valoarea ei a fost adăugată șirului de caractere `uso`, care pe urmă a fost folosit pentru a seta noua variabilă `$PS1`. Observați că la inițializare variabilele nu sunt precedate de `$`.

Directoarele sunt create cu comanda `mkdir(1)`

```
$ pwd
/home/paul/wrk/ub/uso/lab/1
$ mkdir tmp
$ cd tmp
```

Pentru a crea fișiere noi lipsite de conținut folosiți comanda `touch(1)`.

```
$ touch foo
$ ls
foo
```

Argumentul primit este calea către fișierul nou creat. Dacă fișierul deja există, este modificată data la care a fost accesat ultima dată. Operația de copiere se face cu comanda `cp(1)`

```
$ cp foo bar
$ ls
bar foo
```

iar cea de mutare cu comanda `mv(1)`

```
$ mv bar baz
$ ls
baz foo
```

Fișierele se șterg cu comanda `rm(1)`, iar directoarele goale cu comanda `rmdir(1)`

```
$ pwd
/home/paul/wrk/ub/uso/lab/1/tmp
$ ls
baz foo
$ rm baz foo
$ ls
$ cd ..
$ cd ../
$ rmdir tmp
```

Pentru a găsi fișiere se folosește comanda `find(1)`

```
$ find /bin -name ls
/bin/ls
```

unde primul argument este directorul în care să caute și al doilea este numele fișierului. Folosiți `*` pentru a specifica că orice șir de caractere este acceptat mai departe

```
$ find /bin -name l*
/bin/ln
/bin/ls
```

Odată găsite, puteți executa anumite comenzi asupra fișierelor

```
$ mkdir tmp
$ touch tmp/foo tmp/bar tmp/baz
$ ls tmp
bar baz foo
$ find tmp -name b* -exec rm {} \;
$ ls tmp
foo
```

Aici am folosit opțiunea `-exec` a comenzii `find(1)` care execută comanda specificată pentru fiecare rezultat. Notația {} reprezintă rezultatul curent, în cazul nostru întâi `bar` și pe urmă `baz`, iar `\;` semnifică încheierea comenzii `-exec`.

7 Sarcini de laborator

1. Refaceți ierarhia directorului `uso` din Secțiunea 1 folosind comenzile `mkdir(1)` și `touch(1)`.
2. Creați un director `bin` în `$HOME` și adăugați-l în `$PATH`. Copiați un executabil existent și observați cum/dacă se modifică ieșirea comenzii `which(1)` când întrebăm de executabilul copiat. Dacă nu se schimbă, de ce?
3. În ultimul exemplu din Secțiunea 4 `grep(1)` întoarce același număr de rezultate pentru `*` și `?`. Construiți un exemplu nou în care să folosiți cele trei wildcard-uri și să obțineți rezultate diferite.

4. Creați două directoare `orig` și `backup`. În directorul `orig` creați fișierele `foo`,`bar`,`baz`. Folosiți comanda `find(1)` să copiați toate fișierele din `orig` în `backup` dar cu extensia `.orig` în plus (ex. `foo.orig`).

5. Căutați în manualul `rm(1)` cum să ștergeți recursiv și folosiți informația pentru a șterge într-o singură instrucțiune directorul `backup` de mai devreme.